TX-0 COMPUTER
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE 39, MASSACHUSETTS

OBSOLETE

M-5001-19

1 April, 1960

[illegible]

SOME USEFUL MICRO- AND MACRO-INSTRUCTIONS

The instruction vocabulary of MACRO II can be enriched considerably through parameter assignments and macro-instruction definitions. This memorandum is intended to help persons who are relatively new to TX-0 to acquaint themselves with some of its unusual properties (and peculiarities), particularly the microprogramming feature.

Many of the instructions discussed below are not currently defined in MACRO II. However, a tape containing most of these definitions is available from the subroutine library as an English tape (ENG DEFS A) which may be copied into the beginning of the users' English tape. This set of definitions is also available as a binary symbol tape (BN DEFS A) which may be used to restore MACRO before making a conversion. The content of these tapes is given on the last page of this memorandum.

1. Operate Instructions

The following operate instructions not currently defined in MACRO II are frequently useful. Some of these operations are discussed below in more detail.

Instruction	Operation	
anl = opr 305	$AC \cap LR \rightarrow LR$	(Logical Product)
ana = opr 325	$AC \cap LR \rightarrow LR,\ 0 \rightarrow AC$	("AND" and Clear AC)
orl = opr 105	$AC \cup LR \rightarrow LR$	(Logical Sum)
ora = opr 125	$AC \cup LR \rightarrow LR,\ 0 \rightarrow AC$	("OR" and Clear AC)
lcc = lac+com-opr	$-LR \rightarrow AC$	(Place complement of LR in AC)
amz = opr 51	$AC + (-0) \rightarrow AC$	(Add minus zero to AC)
laz = clc+lad-opr	$LR + (-0) \rightarrow AC$	(LR minus zero in AC)
cry = opr 12	$AC \overset{C}{-} LR \rightarrow AC$	(Carry, see below)
lal = cla+cry-opr	$LR + LR \rightarrow AC$	(Equivalent to lac followed by cyl)
ran = cyr+cry-opr		(Random Number Generator, see below)
pen = opr 100		(Set AC Bits 0 and 1 from Light Pen Flip-Flops 1 and 2, respectively, and clear Light Pen Flip-Flops.)

In the TX-0 (and many other machines), addition is accomplished in two steps: partial add and carry. These two operations are also separately useful as operate commands.

1.1 Partial Add

The partial sum of two numbers ($x \wedge y$) is a number each of whose bits represents the "exclusive or" between corresponding bits of x and y. In terms of the Boolean functions "and" ($\cap$), "or" ($\cup$) and "not" ($-$), then,

$$x \wedge y = (x \cap -y) \cup (y \cap -x). \tag{1}$$

In particular, it can be seen that

$$x \wedge 0 = x \tag{2}$$

$$x \wedge x = 0 \tag{3}$$

$$x \wedge -0 = -x \tag{4}$$

Since this operation is associative, it follows from (2) and (3) that

$$x \wedge y \wedge y = x \tag{5}$$

The command lac (LR $\rightarrow$ AC) makes use of relation (2), while lcc (-LR $\rightarrow$ AC) uses (4). lpd (LR $\wedge$ AC $\rightarrow$ AC) can be used to advantage in a number of ways. For example, suppose that it is desired to combine bits 0 - 8 of the AC and bits 9 - 18 of the LR into a single word (the unwanted portions of both registers not assumed to be clear). Making use of (2) and (5), the desired result can be obtained in two steps:

```
lpd
clr+lpd-opr
```

1.2 Carry

The carry operation is loosely analogous to the carry used in pencil-and-paper addition. It may be defined as follows:

> A carry digit is a 1 if, in the next less significant bit, either AC = 0 and MBR = 1, or AC = 1 and the carry digit = 1. The carry digits so determined are partial added to the AC.

Let the result of a carry operation be denoted by AC $\underline{c}$ MBR since (in the one's complement arithmetic of TX-0) the addition operation is the result of the successive execution of partial add and carry, we may write

$$x + y = (x \wedge y) \underline{c}\, y \tag{6}$$

If we define

$$z = x \wedge y$$

it follows from (5) that

$$z \wedge y = x \wedge y \wedge y = x$$

Hence (6) may be written

$$z \underline{c}\, y = (z \wedge y) + y \tag{7}$$

In particular, it may be noted that

$$0 \,\text{Ⓒ}\, y = (0 \wedge y) + y = 2y, \qquad (8)$$

which gives rise to the lal command (LR + LR → AC), and that

$$-y \,\text{Ⓒ}\, y = (-y \wedge y) + y = -0 + y, \qquad (9)$$

which is represented by amz (-0 + AC → AC).

The instruction cry (AC Ⓒ LR → AC) can be used in a number of ways. For example, suppose that it is desired to sense whether bit 9 of the LR contains a 1 or 0. This can be accomplished as follows:

```
cla
add (777377
cry
trn ha
```

It can be seen that if bit 9 of the LR were 0, cry would not change the AC, so control would be transferred to ha. If bit 9 were 1, however, the entire AC would be complemented and the transfer would not be made.

As a second example, suppose that it is desired to cycle left the contents of the LR treating the 12 least significant bits as a unit (i.e., bypassing bits 0 - 5). This is accomplished by

```
cla
add (770000
cry
llr (7777
ana
```

Following this set of operations, the AC and bits 0 - 5 of the LR will be clear, while bits 6 - 17 will be cycled left one position (the former bit 6 going to bit 17).

The instruction ran (= cyr+cry-opr), when used with an appropriate number in the live register, can be used to produce a sequence of psuedo-random numbers. In particular, the instructions

```
cla
add t
llr (355670
ran
sto t
```

can be used repeatedly to generate a sequence of approximately 250,000 random 18-bit words in register t (eventually, a previously computed value is obtained and the sequence repeats). Plus zero is a good choice for the initial contents of t. Minus zero should not be used.

To illustrate another application of ran, suppose that it is desired to make a four-way branch (to locations b0 through b3) dependent on bits 9 and 10 of the LR. Using the same principle that was used before, this is accomplished by

```
cla
add (777377
cry
trn .+4
ran+com-opr
trn b1
tra b0
ran
trn b3
tra b2 .
```

2. Macro-instructions

The following instruction sequences occur often enough in many programs to warrant their definition as macros.

MACRO	INSTRUCTION SEQUENCE
clad A	cla
	add A
llac A	llr A
	lac
llcc A	llr A
	lcc
move A,B	llr A
	slr B
load A,C	llr (C
	slr A
acst C,A	add (C
	sto A
step A,C	cla
	add (C
	add A
	sto A

```
test C,S                                add (-C
                                        trn S

call S                                  llr (tra .+2
                                        tra S

subr S,T                                llr (tra T
                                        tra S
```

2.1 <u>Repetitions</u>

An aspect of programming for the present configuration of TX-0 that differs from many computers is the frequent need for repetition of instructions (e.g. in cycling and shifting). For this purpose, it is convenient to define a set of macros as follows.

```
define    two P      P        P       terminate
define    three P    two P    P       terminate
define    four P     three P  P       terminate
```

etc.

Then, for example, if the programmer wishes to cycle the AC nine places to the right, he can write

nine cyr.

In some cases, the programmer may wish to specify the number of repetitions at some later time by means of a parameter assignment. For example, if he wishes to cycle left zip times ($0 \leq$ zip ≤ 8), he may write

```
loc,        eight cyl
loc+zip|    Next instruction
                 .
                 .
                 .
```

The binary tape will then store cyl in locations loc through loc+8, then read in over this the following instructions, leaving cyl in locations loc through loc+zip-1.

If a large amount of shifting or cycling is required, it is generally more efficient to use closed subroutines together with macros, such as the following:

```
zz=.
define     shrc C    llr (tra .+2
                     tra zz+22-C          terminate
```